JAVANAISE : un cache d'objets répartis en Java

Formation M2PGI (UJF)

Fabienne Boyer
Laboratoire LIG (INRIA-UJF-INPG)
http://membres-liglab.imag.fr/boyer/
Fabienne.Boyer@imag.fr

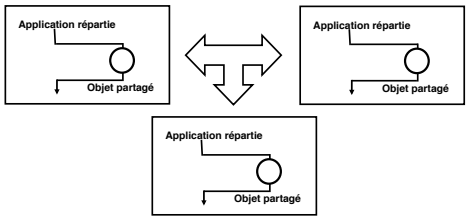
Plan

- Motivations
- Organisation du projet
- API Javanaise
- Gestion de la cohérence
- Planning

Cache d'objets répartis

- **Avantages**
  - Favorise les accès locaux
  - Efficace pour les objets souvent accédés en lecture
  - Efficace pour les objets qui concentrent les phases d'écriture
- **Inconvénients**
  - Complexité de mise en œuvre
  - Complexité de gestion des pannes

Cache d'objets répartis

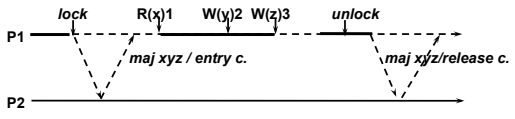


Gestion de la cohérence

- **Cohérence : relation que gardent entre elles les différentes copies d'un objet**
  - Cohérence séquentielle
  - Cohérences faibles
    - cohérence à la sortie (release consistency)
    - cohérence à l'entrée (entry consistency)
  - Cohérences non-séquentielles
    - cohérence causale

Entry vs Release Consistency

- **Cohérence établie sur des points de synchronisation (prise / relâchement/ de verrou)**

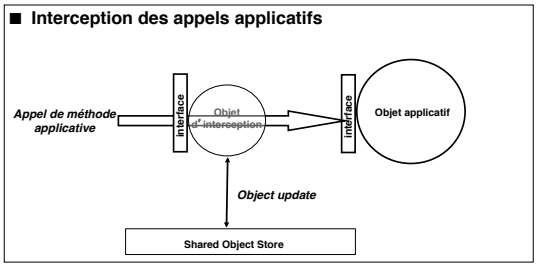


Le projet Javanaise

- **Conception et mise en œuvre d'un service de gestion d'objets Java dupliqués**
  - Au dessus de Java RMI
  - Cohérence de type *entry-consistency*
- **Principe de mise en œuvre**
  - Interception des appels applicatifs sur les objets répartis

Principe de mise en oeuvre

- **Interception des appels applicatifs**

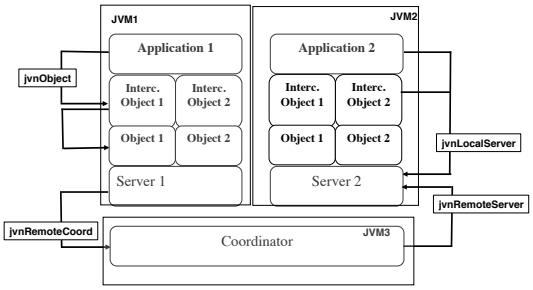


Organisation du projet Javanaise

2 étapes

- JVN1: L'utilisation de Javanaise n'est pas transparente pour le programmeur (usage d'objets d'interception génériques)
- JVN2: Génération automatique d'objets d'interception spécifiques à l'applicatif, afin que le modèle de programmation soit conforme à celui de Java/RMI pour le programmeur
  - Génération à la compilation via l'usage de l'outil Apache/Velocity`
  - Génération à l'exécution via l'usage des dynamic proxies de Java

Architecture Globale



Interface JvnLocalServer

```
// Initialise le serveur JVN (singleton)
   public static jvnServer jvnGetServer();

// Fin d'utilisation du service JVN
  public void jvnTerminate();

// Création d'un objet JVN
public JvnObject jvnCreateObject(Serializable jos);

// Association d'un nom symbolique à un objet JVN
public JvnObject jvnRegisterObject(String jon, JvnObject jo);

// Récupération d'une référence sur un objet JVN à partir de son nom
   symbolique
public JvnObject jvnLookupObject(String jon);
...
```


Interface JvnLocalServer (suite)

```
// Récupère un read lock sur un objet JVN
   public Serializable jvnLockRead(int joi);

// Récupère un write lock sur un objet JVN
   public Serializable jvnLockWrite(int joi);

...
}
```

Interface JvnObject

- **JvnObject : objet d'interception pour un objet partagé**
  - Une référence à un objet partagé pointe sur un JvnObject
- **Interface :**
  - primitives de verrouillage : jvnLockRead(), jvnLockWrite() et jvnUnLock()
- **Implémentation**
  - variable lock (R,W,RC,WC, RWC)
  - *id* unique
  - référence vers l'objet partagé applicatif

Exemple d'utilisation du service Javanaise (JVN 1)

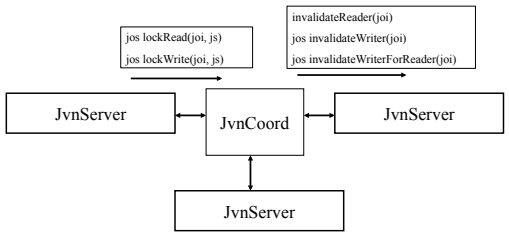
```
// Version 1 de Javanaise :
    JvnServer js = JvnServer.jvnGetServer();
    MyClass o = new MyClass(..);
    JvnObject jvnO = js.jvnCreateObject((..)o);
    …
    jvnO.jvnLockRead();
    // appel d'une méthode applicative
    jvnO.jvnGetObjectState().<method>(<params>);
    jvnO.jvnUnLock();
```

Avec quelques casts…

Gestion de la cohérence

- **Etat du verrou d'un JvnObject :**
  - **NL : no lock**
  - **RC : read lock cached (not currently used)**
  - **WC : write lock cached (not currently used)**
  - **R : read lock taken**
  - **W : write lock taken**
  - **RWC: write lock cached & read taken**

Interactions entre les serveurs et le coordinateur



Interface JvnRemoteServer

```
Interface JvnRemoteServer {
      …
// permet au coord. de réclamer le passage d'un verrou de
   l'écriture à la lecture
      public … jvnInvalidateWriterForReader(int joi);
// permet au coord. de réclamer l'invalidation d'un lecteur
      public … jvnInvalidateReader(int joi);
// permet au coord. de réclamer l'invalidation d'un écrivain
      public … jvnInvalidateWriter(int joi);
   }
```

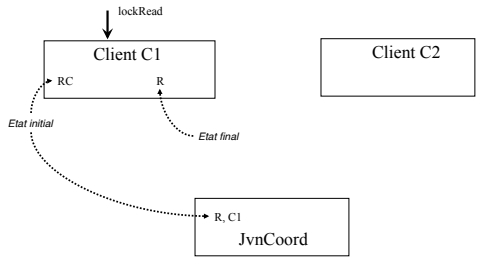

Interface JvnRemoteCoord

```
Interface JvnRemoteCoord {
      …
// permet de réclamer au coord. un verrou en lecture
      public … jvnLockRead(…);
// permet de réclamer au coord. un verrou en écriture
      public … jvnLockWrite(…);
…
   }
```

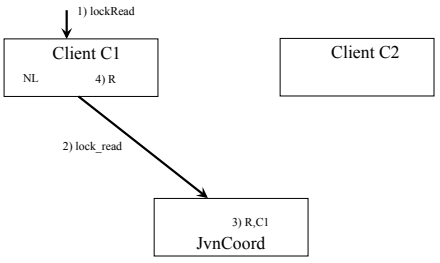

Implémentation du coordinateur

- **Pour chaque objet partagé**
  - Identifiant unique (id)
  - Dernier état validé de l'objet
  - Verrous détenus par les applications clients
- **Le coordinateur contrôle les verrous des applications clientes**
  - Usage de l'interface JvnRemoteServer

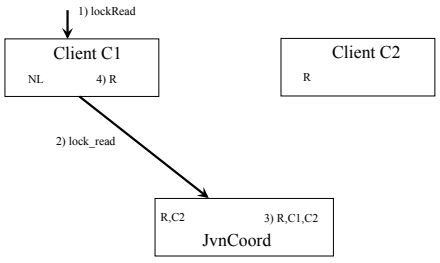
Cohérence : cas de figure 1



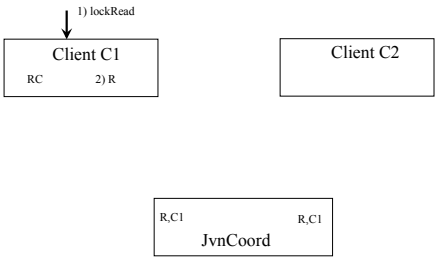
Cohérence : cas de figure 2



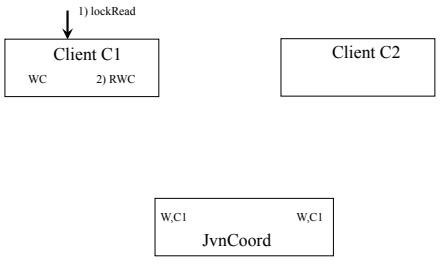
Cohérence : cas de figure 3



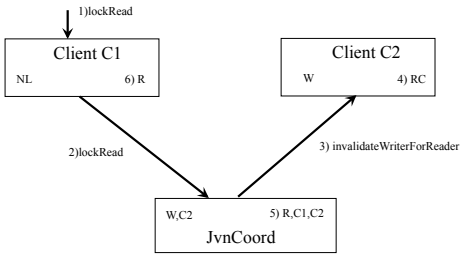
Cohérence : cas de figure 4



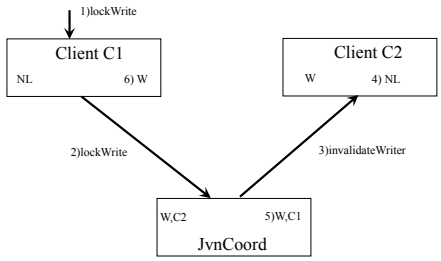
Cohérence : cas de figure 5



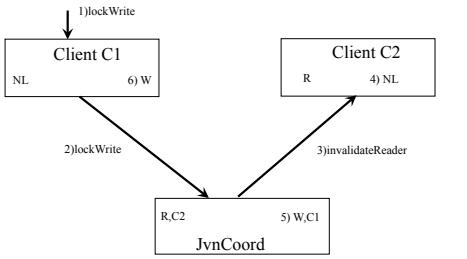
Cohérence : cas de figure 6



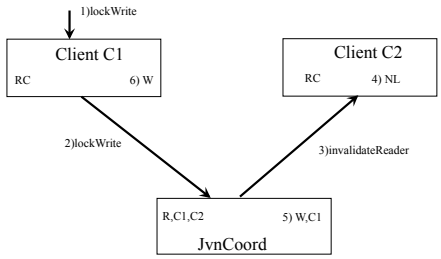
Cohérence : cas de figure 7



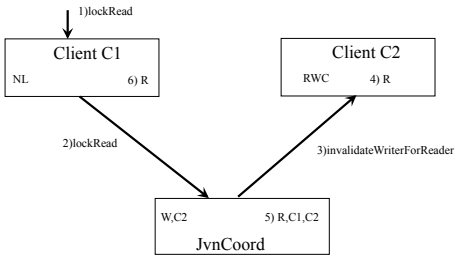
Cohérence : cas de figure 8



Cohérence : cas de figure 9



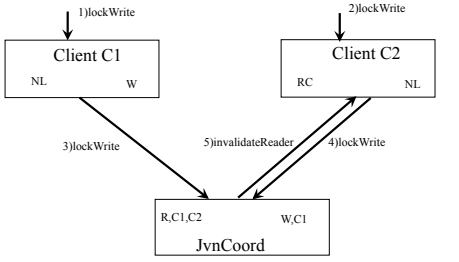
Cohérence : cas de figure 10



Cohérence : principes de mise en œuvre

- **Un objet d'interception (JvnObject) coopère avec le coordinateur pour la gestion de la cohérence**
  - **Il peut être invoqué en parallèle par l'application et par le coordinateur**
  - **Il doit donc être synchronisé**
- **Quand le coordinateur invalide un verrou d'un client**
  - **Ex: invalidateReader(), invalidateWriter(), Invalidate WriterForReader()**
  - **Si le verrou JVN est en cours d'utilisation, il faut attendre**
    - **wait () de Java**
  - **Terminer l'attente lorsque le verrou JVN est libéré**
    - **notify() de Java**

Cohérence : cas critique



Cohérence : cas critique

- Pour éviter le risque d'interblocage
  - **Les méthodes d'invalidation appelée par le coordinateur doivent pouvoir être exécutées à tout moment, même si un appel à lockRead() ou à lockWrite() est en cours**
  - **Donc il faut limiter les portions de code synchronisées dans JvnObject (ne pas englober les appels au coordinateur dans les portions de code synchronisées)**

Etape 2: Javanaise version 2 / JVN2

- **Objectif : augmenter la transparence pour le programmeur**
- **Moyen: utiliser un objet d'interception spécifique, qui fournit les mêmes interfaces que celles de l'objet partagé**
  - **Génération à la compilation de la classe d'interception spécifique en utilisant la réflexivité de Java et l'outil de génération Apache : Velocity**
  - **Génération à l'exécution d'un objet d'interception spécifique dynamique via les *dynamic proxies* de Java**

Exemple d'utilisation de JVN2

```
// Version 2 de Javanaise avec génération à la compilation
    JvnMyClass o = new JvnMyClass(..);
  …
// appel d'une méthode applicative
    o. meth();
```

```
// Version 2 de Javanaise avec génération à l'exécution
(dynamic proxies)
    (MyClass) o = JvnProxy.newInstance(new MyClass())
  …
// appel d'une méthode applicative
    o. meth();
```


Génération à la compilation des objets d'interposition

- **Un objet partagé est d'une classe donnée (ex: *MyClass*)**
- **Analyse des méthodes fournies par cette interface via les fonctions d'introspection Java (java.lang.reflect)**
- **Génération d'une classe d'interception spécifique (*JvnMyClass*), à la place de la classe d'interception générique (*JvnObject*)**
- ***JvnMyClass* implémente les mêmes méthodes / interfaces que *MyClass***
- **Les méthodes fonctionnelles de *JvnMyClass* encapsulent la prise et le relâchement de verrous**
- **Le mode d'accès d'une méthode fonctionnelle est détectable grâce à des annotations Java, ou à noms de méthode spéciaux (…_R ou …_W par ex) ou bien grâce à des étiquettes (tags) sur les méthodes de l'interface.**

Etape 3 : Extensions

- **Extension 1 : gestion de la saturation d'un cache client**
  - **Capacité de « flusher » un ensemble d'objets partagés**
- **Extension 2 : traitement des pannes client**
  - **Tout client peut subir une panne machine à tout instant**
- **Extension 3 : traitement des pannes du coordinateur**
  - **Le coordinateur peut subir une panne machine à tout instant**
  - **On supposera que la machine peut être redémarrée sans perte disque**

Extensions

- **Extension 4 : décentraliser le coordinateur**
  - Toute application utilisant Javanaise participe à la coordination
  - Les agents se synchronisent via un protocole de groupe fiable
- **Extension 5 : invocations transactionnelles**
  - L'objet d'interception fournit des méthodes de démarrage / terminaison de transactions
  - Les méthodes métier invoquées au sein d'une transaction possèdent les propriétés ACID

Extensions

- **Extension 6 : Supporter des clients multi-threadés**
- **Extension 7 : permettre à un objet Javanaise de contenir une référence vers un autre objet Javanaise**
  - Gérer les références contenues dans un objet Javanaise par la spécialisation des primitives de sérialisation / désérialisation

Tests

- **Une application de test fournie (*chat*)**
  - Permet de valider la gestion de la cohérence (verrous) conceptuellement
- **Compléter la base de tests (mode *burst*)**
  - N clients qui stressent le service Javanaise en passant leur temps à acquérir des verrous et à les libérer sur un nombre d'objets restreint
  - Permet de valider la gestion de la cohérence (verrous) du point de vue de l'implémentation
  - Permet de faire des mesures de performance

Planning

Séance 1	Présentation générale du projet **Travail attendu: prototypage JVN1**
Séance 2	Rendu prototypage JVN1 Présentation JVN2 **Travail attendu: finalisation JVN1 et prototypage JVN2**
Séance 3	Rendu prototypage JVN2 Suivi
Séance 4	**Soutenances avec démonstrations**

Evaluation

- Démonstration
- Aboutissement du projet
  - Fonctionnalités
  - Fiabilité
- Tests
- Extensions
- Qualité du code
- Documents associés
- Maîtrise des techniques

A rendre

- Un manuel d'installation et un manuel d'utilisation en anglais
- Les sources du projet en tgz ou en zip

Références

- Modèles à objets dupliqués
  - B.N. Bershad, M.J. Zekauskas, W.A. Sawdon, « The Midway Distributed Shared Memory System », proc. 38th IEEE Computer Society International Conference (COMPCON'93), Feb 93.
  - J.B. Carter, J.K. Benett and W. Zwaenepoel, « Techniques for reducing consistency-related communication in distributed shared memory systems », ACM Transactions on Computer Systems, 13(3), 1995.
  - D. Hagimont, F. Boyer, A Configurable RMI Mechanism for Sharing Distributed Java. Objects, IEEE Internet Computing, Volume 5, numéro 1, Jan 2001.